Modifying objects

Operators and Expressions
```c
float y = 12.5;
int Temperature = 32;
char Letter = 'c';
int Number;
```

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>1002</th>
<th>1003</th>
<th>1004</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>12.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Letter</td>
<td>'c'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ch 3 / Foil 2

Memory Depiction
Assignment Statement

● Basic form
  ● object = expression ;

Celsius = (Fahrenheit - 32) * 5 / 9;
y = m * x + b;
i = i + 1;
Remember = CurrentValue;

● Action
  ■ Expression is evaluated
  ■ Expression value stored in object
Assignment Statement

```java
int NewStudents = 6;
int OldStudents = 21;
int TotalStudents;

TotalStudents = NewStudents + OldStudents;
```

<table>
<thead>
<tr>
<th>NewStudents</th>
<th>OldStudents</th>
<th>TotalStudents</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>21</td>
<td>-</td>
</tr>
</tbody>
</table>

Total Students: 27
Assignment Statement

TotalStudents = NewStudents + OldStudents;

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NewStudents</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>OldStudents</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>TotalStudents</td>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>

OldStudents = TotalStudents;

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NewStudents</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>OldStudents</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>TotalStudents</td>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>
Suppose
Value1 = 10;
Value2 = 20;

Consider
int Hold = Value1;
Value1 = Value2;

Value2 = Hold;
Incrementing

```
int i = 1;

i = i + 1;
```

Assign the value of expression $i + 1$ to $i$

Evaluates to 2
Modifier `const` indicates that an object cannot be changed
  ■ Object is read-only

Useful when defining objects representing physical and mathematical constants

```cpp
const float Pi = 3.1415;
```

Value has a name that can be used throughout the program

```cpp
const int SampleSize = 100;
```

Makes changing the constant easy
  ■ Only need to change the definition and recompile
// Program 3.2

#include <iostream>
#include <string>
using namespace std;

int main() {
    cout << "Enter mass of hydrocarbon (in grams)\n" "followed by the number of carbon atoms\n" "followed by the number of hydrogen atoms\n" "(e.g. 10.5 2 6): " ;

    float Mass;
    int CarbonAtoms;
    int HydrogenAtoms;
    cin >> Mass >> CarbonAtoms >> HydrogenAtoms;
}
// Program 3.2 (continued)

    const int CarbonAMU = 12;
    const int HydrogenAMU = 1;

    long int FormulaWght = (CarbonAtoms * CarbonAMU) + (HydrogenAtoms * HydrogenAMU);

    const double AvogadroNbr = 6.02e23;
    double Molecules = (Mass / FormulaWght) * AvogadroNbr;

    cout << Mass << " grams of a hydrocarbon with " << CarbonAtoms << " carbon atom(s) and " << HydrogenAtoms << " hydrogen atom(s) contains " << Molecules << " molecules" << endl;

    return 0;
}
Sample I/O Behavior

Enter mass of hydrocarbon (in grams) followed by the number of carbon atoms followed by the number of hydrogen atoms (e.g. 10.5 2 6): 19.54 4 30

19.54 grams of a hydrocarbon with 4 carbon atom(s) and 30 hydrogen atom(s) contains 1.50809e+23 molecules
Assignment Conversions

- A floating-point expression assigned to an integer object is truncated
- An integer expression assigned to a floating-point object is converted to a floating-point value
- Consider

```cpp
float y = 2.7;
int i = 15;
int j = 10;
i = y;  // i is now 2
cout << i << endl;
y = j;  // y is now 10.0
cout << y << endl;
```
Compound Assignment

- C++ has a large set of operators for applying an operation to an object and then storing the result back into the object
- Examples

```cpp
int i = 3;
i += 4; // i is now 7
cout << i << endl;

float a = 3.2;
a *= 2.0; // a is now 6.4
cout << a << endl;
```
C++ has special operators for incrementing or decrementing an object by one

Examples

```cpp
int k = 4;
++k;       // k is 5
k++;       // k is 6
cout << k << endl;
int i = k++; // i is 6, k is 7
cout << i << " " << k << endl;
int j = ++k; // j is 8, k is 8
cout << j << " " << k << endl;
```
Nonfundamental Types

- Nonfundamental as they are additions
- C++ permits definition of new types and *classes*
  - A class is a special kind of type
- Class objects typically have
  - *Data members* that represent attributes and values
  - *Member functions* for object inspection and manipulation
  - Members are accessed using the selection operator (.)
    
    ```
    j = s.size();
    ```
  - *Auxiliary* functions for other behaviors
- Libraries often provide special-purpose types and classes
- Programmers can also define their own types and classes
Nonfundamental Types

- Examples
  - Standard Template Library (STL) provides class `string`
  - EzWindows library provides some graphical types and classes
    - `SimpleWindow` is a class for creating and manipulating window objects
    - `RectangleShape` is a class for creating and manipulating rectangle objects
Nonfundamental Types

- To access a library use a preprocessor directive to add its definitions to your program file
  
  ```
  #include <string>
  ```

- Using statement makes syntax less clumsy

  - Without it
    ```
    std::string s = "Wahoo";
    std::string t = "Spiffy";
    ```

  - With it
    ```
    using namespace std; // std contains string
    string s = "Wahoo";
    string t = "Spiffy";
    ```
Class string

- Class string
  - Used to represent a sequence of characters as a single object
- Some definitions
  ```c#
  string Name = "Joanne";
  string DecimalPoint = ".";
  string Question = '?'; // illegal
  ```
Class string

- Some string member functions
  - `size()` determines number of characters in the string
    ```
    string Saying = "Rust never sleeps.";
    cout << Saying.size() << endl;        // 18
    ```
  - `substr()` determines a substring (Note first position has index 0)
    ```
    string Word = Saying.substr(11, 16);  // sleeps
    ```
  - `find()` computes the position of a subsequence
    ```
    int j = Word.find("ee");            // 2
    int k = Rust.find("steel");        // ?
    ```
Class string

- Auxiliary functions and operators
  - getline() extracts the next input line
    ```
    string Response;
    cout << "Enter text: ";
    getline(cin, Response, '\n');
    cout << "Response is " << Response
          << "\"" << endl;
    ```
  - Example run
    Enter text: Want what you do
    Response is "Want what you do"
Class string

- Auxiliary operators
  - + string concatenation
    ```
    string Part1 = "Me";
    string Part2 = " and ";
    string Part3 = "You";
    string All = Part1 + Part2 + Part3;
    ```
  - += compound concatenation assignment
    ```
    string ThePlace = "Brooklyn";
    ThePlace += ", NY";
    ```
```cpp
#include <iostream> // Program 3.4
#include <string>
using namespace std;

int main() {
    cout << "Enter the date in American format: "
         << "(e.g., December 29, 1953) : " << endl;
    string Date;
    getline(cin, Date, '\n');
    int i = Date.find(" ");
    string Month = Date.substr(0, i);
    int k = Date.find(",", i);
    string Day = Date.substr(i + 1, k - i - 1);
    string Year = Date.substr(k + 2, Date.size() - 1);
    string NewDate = Day + " " + Month + " " + Year;
    cout << "Original date: " << Date << endl;
    cout << "Converted date: " << NewDate << endl;
    return 0;
}
```
EzWindows Library Objects

- Definitions are the same form as other objects
- Example
  
  ```cpp
  SimpleWindow W;
  ```
- Most non-fundamental classes have been created so that an object is automatically initialized to a sensible value
- `SimpleWindow` objects have member functions to process messages to manipulate the objects
- Most important member function is `Open()` which causes the object to be displayed on the screen
- Example
  
  ```cpp
  W.Open();
  ```
Initialization

- Non-fundamental objects may have several attributes to initialize
- Syntax for initializing an object with multiple attributes
  \[
  \text{Type Identifier(Exp}_1, \ Exp_2, \ldots, \ Exp_n);\]
- SimpleWindow definitions can optionally specify attributes
  \[
  \text{SimpleWindow W("Window Fun", 8, 4);}\]
  - First attribute
    - Window banner
  - Second attribute
    - Width of window in centimeters
  - Third attribute
    - Height of window in centimeters
An EzWindows Program

```cpp
#include <iostream>
#include <string>
using namespace std;
#include "ezwin.h"
int ApiMain() {
    SimpleWindow W("A Window", 12, 12);
    W.Open();

    cout << "Enter a character to exit" << endl;
    char a;
    cin >> a;

    return 0;
}
```
Sample Display Behavior
RectangleShape Objects

- EzWindows library also provides `RectangleShape` class for manipulating rectangles
- `RectangleShape` objects can specify the following attributes
  - A `SimpleWindow` object that contains the rectangle (mandatory)
  - Offset from left edge of the `SimpleWindow`
  - Offset from top edge of the `SimpleWindow`
    - Offsets are measured in centimeters from rectangle center
  - Width in centimeters
  - Height in centimeters
  - Color
    - `color` is an EzWindows type
RectangleShape Objects

- Examples
  
  SimpleWindow W1("My Window", 20, 20);
  SimpleWindow W2("My Other Window", 15, 10);

  RectangleShape R(W1, 4, 2, Blue, 3, 2);
  RectangleShape S(W2, 5, 2, Red, 1, 1);
  RectangleShape T(W1, 3, 1, Black, 4, 5);
  RectangleShape U(W1, 4, 9);
RectangleShape Objects

- **Major** RectangleShape member functions for processing messages
  - **Draw()**
    - Causes rectangle to be displayed in its associated window
  - **GetWidth()**
    - Returns width of object in centimeters
  - **GetHeight()**
    - Returns height of object in centimeters
  - **SetSize()**
    - Takes two attributes -- a width and height -- that are used to reset dimensions of the rectangle
Another EzWindows Program

```cpp
#include <iostream>
#include <string>
using namespace std;
#include "rect.h"

int ApiMain() {
    SimpleWindow W("Rectangular Fun", 12, 12);
    W.Open();
    RectangleShape R(W, 5.0, 2.5, Blue);
    R.Draw();
    cout << "Enter a character to exit" << endl;
    char Response;
    cin >> Response;
    return 0;
}
```
Sample Display Behavior